Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Million Atom KS-DFT with CP2K

Iain Bethune[a]*, Adam Carter[a], Xu Guo[a], Paschalis Korosoglou[b,c]

[a]*EPCC, The University of Edinburgh, James Clerk Maxwell Building, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom*
[b]*AUTH, Aristotle University of Thessaloniki, Thessaloniki 52124, Greece,*
[c]*GRNET, Greek Research & Technology Network, L. Mesogeion 56, Athens 11527, Greece*

Abstract

CP2K is a powerful materials science and computational chemistry code and is widely used by research groups across Europe and beyond. The recent addition of a linear scaling KS-DFT method within the code has made it possible to simulate systems of an unprecedented size – 1,000,000 atoms or more – making full used of Petascale computing resources. Here we report on work undertaken within PRACE 1-IP WP 7.1 to port and test CP2K on Jugene, the PRACE Tier 0 BlueGene/P system. In addition, development work was performed to reduce the memory usage of a key data structure within the code, to make it more suitable for the limited memory environment of the BlueGene/P. Finally we present a set of benchmark results and analysis of a large test system.

Project ID: PRA1IC

1. Project Overview

"CP2K is a freely available (GPL) program, written in Fortran 95, to perform atomistic and molecular simulations of solid state, liquid, molecular and biological systems" [1]. Of the wide range of methods available in CP2K, for this project we are concerned with the recently implemented linear scaling Kohn-Sham Density Functional Theory (KS-DFT) calculations. This method achieves linear scaling (in the number of atoms) by the use of a massively parallel sparse matrix library DBCSR, which has been designed by the CP2K developers specifically for atomistic simulation. Using DBCSR, all $O(N^3)$ dense linear algebra operations are replaced with sparse O(N) counterparts, thus avoiding expensive matrix diagonalisation, the bottleneck of traditional DFT calculations.

These developments will allow the simulation of very large systems of the order of 1,000,000 atoms to be simulated efficiently, given enough CPU resources. For example, calculations of 100,000 atoms have been performed on JaguarPF, using 1000 or more nodes (12,000 cores). Enabling such large-scale simulation with CP2K will give a step-change in capability to PRACE users, allowing truly nanoscale systems, such as small virii (e.g. the Satellite Tobacco Mosaic Virus - STMV), or novel nanotransistors to be simulated in their entirety.

In order to enable the above for European researchers, this project aims to port and verify the correctness of CP2K on Jugene, the BlueGene/P PRACE Tier 0 system at Jülich. In particular, we are concerned with the mixed-

* Corresponding author. *E-mail address*: ibethune@epcc.ed.ac.uk

mode MPI/OpenMP variant of the code, which has not to our knowledge been tested before on this platform. In addition, modifications have been made to the code to reduce the memory footprint of important data structures. While most of the data structures within CP2K (grids, matrices) are distributed, there remains some information relating to atoms (basis, coordinates, topology), which is not. Initial testing on a system of around a million atoms has indicated that this amounts to approximately 1GB of data per MPI process, hence the need for a mixed-mode approach, allowing use of the full 2GB memory of a BlueGene/P node and maintaining good performance. We will also benchmark the performance of CP2K on Jugene using a range of large systems, including the STMV system.

2. Porting CP2K to the BlueGene/P

2.1. Compilation

CP2K is currently available either directly through CVS or by downloading nightly snapshots through ftp. On the login node of the BlueGene/P system (Jugene) at Juelich (FZJ) we have used directly the CVS repository for obtaining and updating the source code.

The first step towards porting the CP2K application on Jugene has been the compilation and linking of the libint library. The libint library is comprised of C/C++ functions for the efficient evaluation of several kinds of two-body molecular integrals. With respect to CP2K building and linking the libint library was required, as the HFX modules use it in order to calculate the four center electron repulsion integrals.

Due to the fact that the CP2K Fortran code has to inter-operate with the libint C/C++ function prototypes, linking libint to the CP2K binaries has been troublesome as under several circumstances this stage failed. Specifically, compiler errors where produced when linking library files of the libint library that were built using the IBM XL family of compilers. In order to bypass these errors, we had to use the GCC family of compilers for the compilation of the libint library. Linking thereafter with the CP2K binaries required one additional step, which was to build a C++ wrapper file that is provided directly by the CP2K distribution and to include the resulting object file during the final link stage. The GCC suite was used for compiling the wrapper source file.

To build the CP2K application an architecture (arch) file is used. This file is tied to the underlying system that CP2K is being built for and it is used to initialize variables within a generic Makefile. To build the MPI only version of CP2K on Jugene the arch file shown in listing 1 was used.

The compiler used for building and linking CP2K is the MPI enabled wrapper of the IBM XL Fortran compiler for BlueGene/P (version 11.1). We have used the thread safe version of the compiler for Fortran 95 source files. As is evident from the arch file, the external libraries that have been used for building CP2K besides libint are:

- ESSL,
- FFTW,
- LAPACK,
- BLACS and
- ScaLAPACK

The stability and the performance of the binary that we have built using this arch file was initially tested by submitting batch jobs using several sample input files. The optimization flags related to the compiler that we used in order to build a pure MPI version were "`-O2 -qarch=450d -qtune=450 -qessl`". This binary has been tested using the regtest suite provided by the CP2K distribution (see section 4).

For building a stable hybrid MPI/OpenMP version of CP2K the arch file shown in listing 2 was used.

```
FFTW3_BASE = /bgsys/local/fftw3

CC       = xlc
CPP      = cpp -traditional -E
FC       = mpixlf95_r -qsuffix=f=f90
LD       = mpixlf95_r
AR       = ar -r

DFLAGS   = -D__HAS_NO_ISO_C_BINDING -D__AIX -D__ESSL -D__FFTSG -D__parallel \
           -D__BLACS -D__SCALAPACK -D__LIBINT -D__FFTW3
CPPFLAGS = -C $(DFLAGS) -P -I$(FFTW3_BASE)/include
FCFLAGS  = -O2 -qarch=450d -qtune=450 -qstrict -qessl -Wl,--relax -qcompact \
  -qextname=wrapper_build_deriv1_eri:wrapper_build_eri:wrapper_free_libderiv:wrapper_free_libint:
wrapper_init_deriv:wrapper_init_lib
FCFLAGS2 = -O0 -qarch=450d -qtune=450 -qessl \
  -qextname=wrapper_build_deriv1_eri:wrapper_build_eri:wrapper_free_libderiv:wrapper_free_libint:
wrapper_init_deriv:wrapper_init_lib
LDFLAGS  = $(FCFLAGS)
LIBS     = /bgsys/local/scalapack/lib/libscalapack.a \
           /bgsys/local/blacs/lib/libblacsF77init.a \
           /bgsys/local/blacs/lib/libblacs.a \
           /bgsys/local/lapack/lib/liblapack.a \
           $(FFTW3_BASE)/lib/libfftw3.a \
           /opt/ibmmath/essl/4.4/lib/libesslbg.a \
           /homeb/pra1ic/pra1ic03/cp2k/tools/hfx_tools/libint_tools/libint_cpp_wrapper.o \
           /homeb/pra1ic/pra1ic03/lib/gnu/libint/lib/libderiv.a \
           /homeb/pra1ic/pra1ic03/lib/gnu/libint/lib/libint.a \
           -lstdc++

OBJECTS_ARCHITECTURE = machine_aix.o

bibliography.o: bibliography.F
       $(FC) -c $(FCFLAGS2) $<
```

Listing 1 - Arch file used to build pure MPI version of CP2K on Jugene

Notice that the SMP flag used to instruct the compiler to translate the OpenMP source code directives is "`-qsmp=omp:noopt:noauto`", which effectively disables the optimizations on the parallelized program code along with the automatic parallelization. Apart from these options, "`-qnohot`" is used in order to disable any high order compiler driven transformations. It is also worth noting that when we used the SMP version of the ESSL library, the binary that was produced was unstable and lead to crashes using several of the example cases from the regtest suite. One last point to consider is the usage of the "`-D__HAS_NO_OMP_3`" in the preprocessor stage, which is needed to avoid compilation errors during the building phase, as OpenMP 3.0 is not supported by the IBM XL compilers (version 11.1) that have been used. Using these options combined a stable hybrid binary containing both MPI calls and OpenMP directives has been built. This binary has been tested using the regtest suite provided by the CP2K distribution (see section 4).

```
FFTW3_BASE = /bgsys/local/fftw3

CC       = xlc_r
CPP      = cpp -traditional -E
FC       = mpixlf95_r -qsuffix=f=f90
LD       = mpixlf95_r
AR       = ar -r
DFLAGS   = -D__HAS_NO_ISO_C_BINDING -D__AIX -D__ESSL -D__FFTSG -D__parallel \
           -D__BLACS -D__SCALAPACK -D__LIBINT -D__HAS_NO_OMP_3 -D__FFTW3
CPPFLAGS = -C $(DFLAGS) -P -I$(FFTW3_BASE)/include
FCFLAGS  = -qsmp=omp:noopt:noauto -qthreaded -qnohot \
           -O2 -qarch=450d -qtune=450 -qstrict -qessl -Wl,--relax -qcompact \
  -qextname=wrapper_build_deriv1_eri:wrapper_build_eri:wrapper_free_libderiv:wrapper_free_libint:
wrapper_init_deriv:wrapper_init_lib
FCFLAGS2 = -qsmp=omp:noopt:noauto -qthreaded -qnohot \
           -O0 -qarch=450d -qtune=450 -qessl \
  -qextname=wrapper_build_deriv1_eri:wrapper_build_eri:wrapper_free_libderiv:wrapper_free_libint:
wrapper_init_deriv:wrapper_init_lib
LDFLAGS  = $(FCFLAGS)
LIBS     = get_memory.o /bgsys/local/scalapack/lib/libscalapack.a \
           /bgsys/local/blacs/lib/libblacsF77init.a \
           /bgsys/local/blacs/lib/libblacs.a \
           /bgsys/local/lapack/lib/liblapack.a \
           $(FFTW3_BASE)/lib/libfftw3.a \
           /opt/ibmmath/essl/4.4/lib/libesslbg.a \
           /homeb/pra1ic/pra1ic03/cp2k/tools/hfx_tools/libint_tools/libint_cpp_wrapper.o \
          /homeb/pra1ic/pra1ic03/lib/gnu/libint/lib/libderiv.a \
          /homeb/pra1ic/pra1ic03/lib/gnu/libint/lib/libint.a \
           -lstdc++

OBJECTS_ARCHITECTURE = machine_aix.o

bibliography.o: bibliography.F
	$(FC) -c $(FCFLAGS2) $<
```

Listing 2 - Arch file for building the hybrid MPI+OpenMP version of CP2K on Jugene

2.2. Memory reporting

CP2K contains a module (`machine.F`) containing generic interfaces to system-specific functionality including timing, process control, and memory usage among others. Which particular implementation of these functions to use is selected via an architecture-specific preprocessor macro defined in the arch file. In this case `-D__AIX` compiles the IBM-specific machine file. However, after running the code, and seeing that the reported memory usage was zero, it was discovered that the relevant `m_memory` function (which should return the total memory size of the process) was in fact only a stub implementation which returned zero. Following the advice on the Jugene FAQ [2] that this information could be obtained via the `getrusage()` function call, a Fortran interface and types were defined, using the `ISO_C_BINDING` module, allowing this C routine to be called directly from within CP2K. This code (see Listing 3) has been added to the main CP2K CVS repository and is now publicly available.

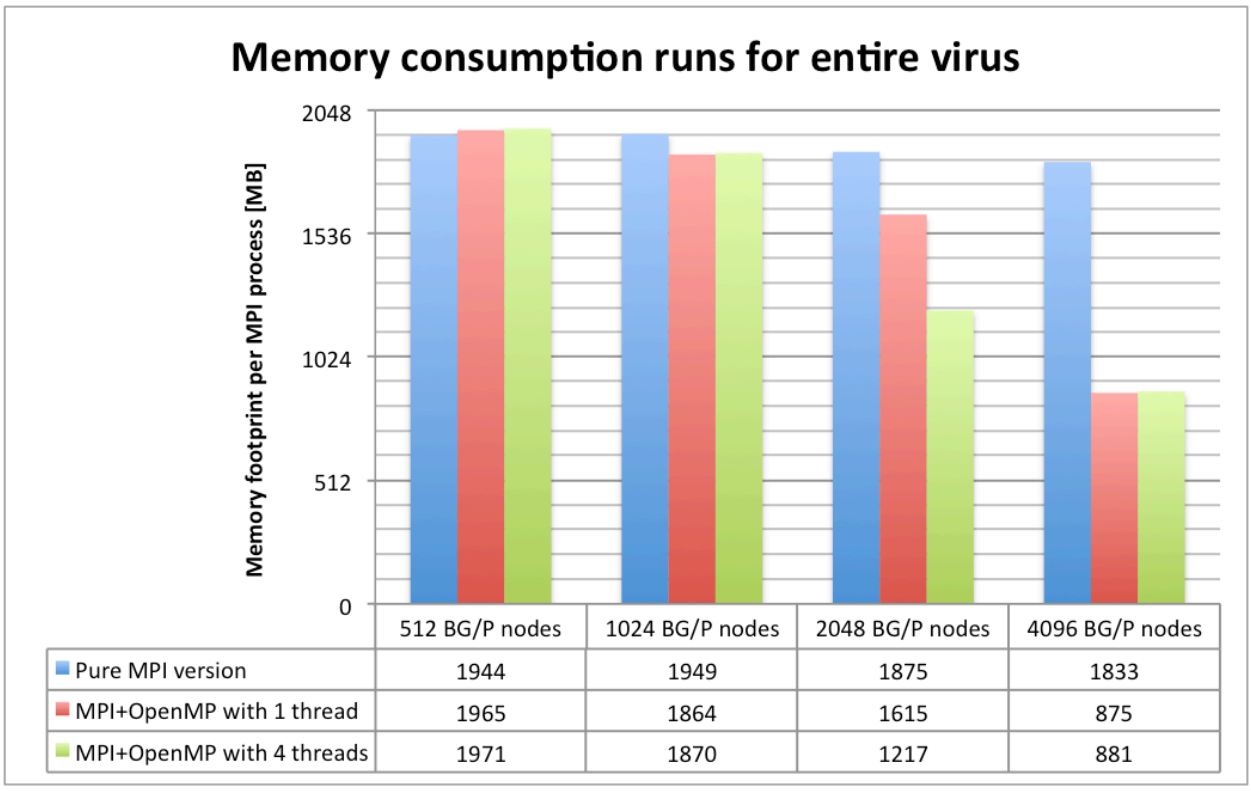
```
MODULE machine_aix
  USE ISO_C_BINDING
…
 TYPE, BIND(C) :: timeval
     INTEGER(C_LONG) :: tv_sec, tv_usec
  END TYPE

  TYPE, BIND(C) :: rusage
     TYPE(timeval) :: ru_utime, ru_stime
     INTEGER(C_LONG) :: ru_maxrss, ru_ixrss, ru_idrss, ru_isrss, ru_minflt, ru_majflt, &
                        ru_nswap, ru_inblock, ru_oublock, ru_msgsnd, ru_msgrcv, &
                        ru_nsignals, ru_nvcsw, ru_nivcsw
  END TYPE

  INTERFACE    INTEGER(C_INT) FUNCTION getrusage (who, rusage) BIND(C, name='getrusage')
    USE ISO_C_BINDING
      INTEGER(C_INT), VALUE :: who
      TYPE(C_PTR),    VALUE :: rusage
    END FUNCTION getrusage
  END INTERFACE
…
! returns the total amount of memory [bytes] in use, if known, zero otherwise
! **************************************************************************
  FUNCTION m_memory()
    INTEGER(KIND=int_8)                     :: m_memory

    INTEGER(C_INT)                          :: ret
    TYPE(rusage), TARGET                    :: usage

    ret = getrusage(0, C_LOC(usage))
    m_memory = usage%ru_maxrss * 1024
  END FUNCTION m_memory
…
END MODULE machine_aix
```

Listing 3 – Code for reporting current memory usage on BlueGene and other IBM systems

Using this modification and prior to implementing the memory reduction modifications (described in section 3) we have measured the memory consumption using the entire virus (STMV) modelled using Semi-Empirical (SE) force evaluation on a different BG/P partition sizes. There will be clearly some difference in the memory required for the SE method and full DFT, however, SE was chosen at this stage as it typically results in shorter job times. For completeness we have used both versions of CP2K (pure MPI and mixed-mode MPI/OpenMP) varying also the number of OpenMP threads between 1 and 4 in the latter case. Our results on memory resources consumed per MPI process are given in Figure 1.

In order to reduce the CPU time consumption on JUGENE we have limited the maximum allowed wall clock time per job to 30 minutes. Of these runs only the pure MPI version of CP2K running on 512 BG/P nodes clearly terminated due to insufficient memory resources. In all other cases execution stopped due to the enforced wall clock limitation. It is worth mentioning that in none of these runs did the execution reach the main calculation SCF phase. Thus, we suspect that jobs where the reported memory consumption was close to 2GB per MPI process would also

suffer from insufficient memory resources if we had enforced a higher wall clock limitation or allowed them to run to completion.



	512 BG/P nodes	1024 BG/P nodes	2048 BG/P nodes	4096 BG/P nodes
Pure MPI version	1944	1949	1875	1833
MPI+OpenMP with 1 thread	1965	1864	1615	875
MPI+OpenMP with 4 threads	1971	1870	1217	881

Figure 1 - Results on memory consumption per MPI process for STMV-SE case

3. Reducing memory usage for large systems

Our goal was to reduce the memory usage of the program for large systems since this can be a limiting factor in the size of systems that can be simulated, particularly on low-memory nodes such as on the BlueGene/P.

Our strategy was to replace the linked list data structure used for particle neighbor lists and replace it with an array data structure. The rationale for this is that each node in a linked list also contains a pointer to the next item in the list. Replacing these structures with arrays means that the items can be packed next to each other in memory removing the need for the pointers, and thus saving space. The list item in question is 40 bytes per node, plus the 8 byte pointer, so our target was a 17% reduction in memory usage for the neighbor lists. In practice, the picture is not quite so clear cut, as the algorithms required, for example, to add a particle to a list differ between the two implementations, and result in a different pattern of memory allocations.

3.1. Process

The code was examined and its structure was briefly discussed with the developers of the code. Whilst the code is written in a modular way, to a developer new to the code it appears quite complex. A "dummy" code with considerably less complexity was therefore written using linked lists on which to test the refactoring approach. This dummy code was refactored to replace the linked list data structure with an array data structure and the refactoring strategy was evaluated. It was determined that the same strategy could be applied to the main code.

The main code was refactored using the same strategy. In summary, this consisted of introducing a new "payload" type containing all of the components of the list item type. Getter and setter functions were introduced for these components. The correctness of the code was checked by comparing its output to that of the original version (specifically, the contents of the `*.ener` files were compared by eye). A list of payloads could then be replaced with an array of payloads to complete the change of data structures. The correctness of the code was rechecked. The codes can be seen to give the same answers, and will be thoroughly compared using CP2K's set of regression tests before checking the updated code back into the project's CVS repository.

Memory use was measured using the code's own memory profiling routines called from the `timeset` and `timestop` subroutines from the module timings in file `timings.F`. These measure memory use at the resolution of a page (4KB), which is useful for determining the size of simulation that will fit in memory, but they do have to be interpreted carefully during memory profiling and debugging as it is not possible to observe the effects of each individual memory allocation and deallocation.

All development and the memory usage tests were undertaken on the machine Palu (Cray XE6) at CSCS.

3.2. Implementation Notes

Arrays are originally allocated to be small in size, and if an element needs to be added to an array that is full, a new array of double the size is created and the contents of the smaller array are copied into the larger one (note that this potentially reduces the speed of the program, and this trade-off should ideally be evaluated). This gives an amortized O(1) cost of adding an element to the array, in common with the link list method. At the end of the point where lists are built, the arrays are trimmed back to their optimum size (containing no empty elements) with a similar copying of elements to new smaller arrays.

3.2.1. Modifications

Due to the complexity of the code, it was not initially clear how much of the code would need to be modified. The modular nature of the code, however, meant that in the end, the only files which required modification were `qs_neighbor_lists.F` and `qs_neighbor_list_types.F`. The modifications are summarised below:

- `qs_neighbor_lists.F`
  - `SUBROUTINE` **`build_neighbor_lists`** (changed)
    - Added a call to `trim_neighbor_lists` towards the end of the subroutine, after each set of lists has been built
- `qs_neighbor_list_types.F`
  - `TYPE` **`payload`** (added)
    - Arrays of list nodes are arrays of this type
    - This payload contains the variables *`r`*, *`cell`*, and *`neighbor`* from the original `neighbor_node_type`
  - `TYPE` **`neighbor_node_type`** (removed)
    - Replaced by payload type
  - `TYPE` **`neighbor_list_iterator_type`**
    - Integer `nni` now identifies the neighbor node instead of the pointer `neighbor_node`
  - `SUBROUTINE` **`grow_array`** (added)
    - Called by `add_neighbor_node` when the existing array is not big enough to hold the new node
  - `SUBROUTINE` **`trim_neighbor_list`** (added)

- Called after a neighbor list has been built to ensure that the array is no bigger than it needs to be to hold the list

- Setters and getters for `r`, `cell`, and `neighbor` (added)
  - `FUNCTION` **`get_r`**
  - `FUNCTION` **`get_cell`**
  - `FUNCTION` **`get_neighbor`**
  - `FUNCTION` **`set_r`**
  - `FUNCTION` **`set_cell`**
  - `FUNCTION` **`set_neighbor`**
  - These were added so that the rest of the code need not be aware of how a list node is implemented.
- `SUBROUTINE` **`neighbor_list_iterator_create`** (changed)
- `SUBROUTINE` **`nl_set_sub_iterator`** (changed)
- `FUNCTION` **`neighbor_list_iterate`** (changed)
- `FUNCTION` **`nl_sub_iterate`** (changed)
  - Modified in several places to use an integer rather than a pointer to track the current iterator position
- `SUBROUTINE` **`get_iterator_info`** (changed)
  - The call to `get_neighbor_node` has been modified to work with the new data types
- `SUBROUTINE` **`add_neighbor_list`** (changed)
  - Changes made so that new neighbor lists are created with the new data types
- `SUBROUTINE` **`add_neighbor_node`** (changed)
  - Probably the most-modified part of the code
  - Nodes are now added into an existing array rather than adding nodes to a linked list. If the array is not big enough, it is increased in size with a call to `grow_array`.
- `SUBROUTINE` **`deallocate_neighbor_list`** (changed)
  - Changed to deallocated arrays instead of linked lists
- `FUNCTION` **`first_node`** (removed)
  - Not needed in the new implementation
- `SUBROUTINE` **`get_neighbor_node`** (changed)
  - Now returns a payload instead of a pointer to a node
  - Modified to use getters and setters

3.3. Results

Unfortunately, at the time of writing, it appears that these modifications have not reduced the total memory use of the code. The results that follow illustrate the effect of making the changes from a linked list to an array-based data structure. The new version of the code runs successfully on those benchmarks on which it has been tested (H2O-64 and STMV) and gives the same results as the older version.

Investigations have been undertaken as to why the new version of the code does not improve memory performance and the reason appears to be that whilst arrays take up less space, they are less versatile and adding items to an array can sometimes require that the array be grown and/or trimmed. To implement this in FORTRAN 95 actually requires an array copy and the deallocation of the old array. The effect of this is that gaps can be left in memory that cannot be filled unless sufficiently small new arrays are allocated to fill in the gaps – essentially memory fragmentation. For this reason that improvements in memory consumption are not uniform and will potentially depend on problem size and other input parameters.

A summary of the main results obtained follows in table 1 and figure 2. Since runs of the code with realistically large datasets take some time to run, not all of the runs ran to the end of the program, however it is possible to observe the memory on exit from the `build_qs_neighbor_lists` subroutine, and it is this figure that is used in the table below which shows the results for a few different runs.

Table 1 Total memory usage reported after `build_qs_neighbor_lists`

Benchmark	**Number of Processes**	**Code Version**	**Memory Use**
STMV	200	Linked-list-based particle lists	4988MB
		Array-based particle lists	6270MB
H2O-64	100	Linked-list-based particle lists	3254MB
		Array-based particle lists	3254MB

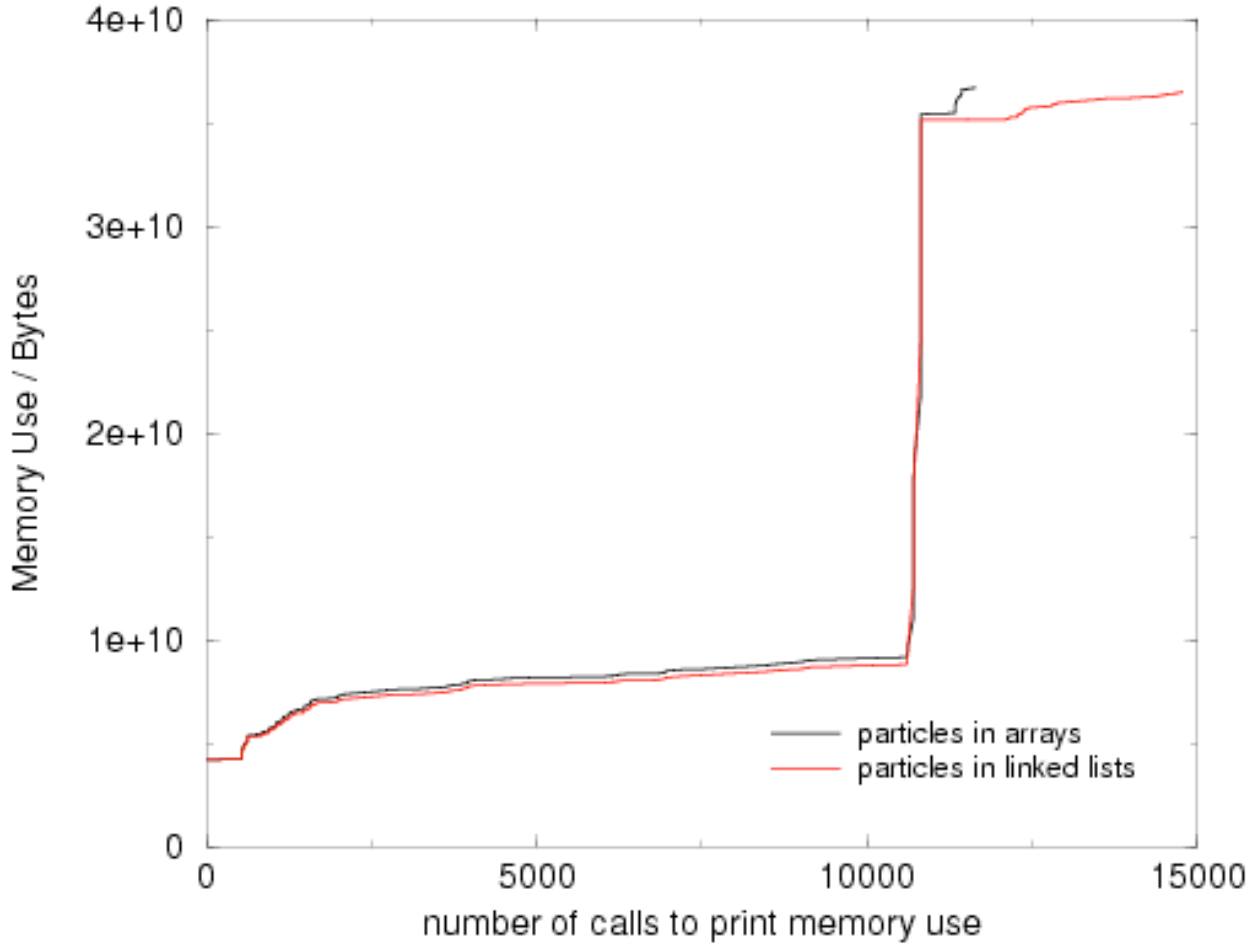


Figure 2 Comparing memory allocate of linked list and array versions of the neighbour lists

4. Regression testing & benchmark results

Initial regression test and benchmark results were all generated with CP2K version 2.2.177 (CVS head from 30th March 2011). Ongoing development of DBCSR has introduced significant changes since then which affects both the performance and compiler-friendliness of the code, in particular overcoming some issues with caused the mixed-mode MPI/OpenMP version of the code. Therefore the mixed-mode regression test results reported are from CP2K version 2.2.228 (20th May 2011).

4.1. Regression testing

The CP2K distribution provides a set of regression test inputs to help verify the correctness of the existing functionalities of the code. There are total 2163 regression test inputs, distributed in 99 directories which are listed in the TEST_DIRS file. In this project, all the regression tests were run on the PRACE BG/P system, Jugene, using the executable of CP2K MPI version building and the executable of CP2K Mixed-mode (MPI/OpenMP) version building respectively.

Several limitations should be noted when running the CP2K regression tests on JUGENE:

- Limited memory size

Each CP2K regression test can only run on one processor. Some tests appear to require large memory for execution. To use the maximum memory size available for each processor on the JUGENE nodes, all the regression tests used SMP mode and the actual available memory was slightly less than 2GB on each node.

- Maximum execution time

For small sized jobs (<256 nodes), the maximum execution time is 30 minutes for each job on Jugene. Due to this maximum execution time limit, the CP2K regression tests were run separately for each test directory. This enables most of the regression tests to be completed within 30 minutes, but 30 minutes is still not enough for some of the regression tests directories which requires longer total time.

Table 2 shows the CP2K regression tests status running with the MPI-only version of the executable. Further detailed regression test status can be found at the Appendix. 1723 regression tests passed when running with the MPI version executable, 48 failed and 392 were missed. The main reasons for the failed tests are apparently due to memory limitations where the failed executions were killed by the signal 16 or signal 6. The missed tests have two possible reasons: some tests have no output content and the probable reason for this is the execution time limit on JUGENE, i.e. the regression tests haven't finished when reaching the 30 minutes time limit; and some existing test directories are not listed in the TEST_DIR documents and therefore are not able to be tested individually.

Table 2 CP2K regression tests on JUGENE running with MPI version executable (CP2K 2.2.177)

	Count	Rate
Passed	17123	79.7%
Failed	48	2.2%
Missed	392	18.1%

The later version CP2K 2.2.228 contains several fixes and improvements to the OpenMP implemented within the DBCSR library. Table 3 shows the CP2K regression tests status running with the mixed mode MPI/OpenMP

version of the executable. Further detailed regression test status is listed in the Appendix. 1325 tests were passed, 18 failed, 818 tests has no output and 2 test produced wrong answers (compared to version 2.2.177)

Table 3 CP2K regression tests on JUGENE running with Mixed-mode version executable (CP2K 2.2.228)

	Count	Rate
Passed	1325	61.3%
Failed	18	0.8%
Missed	818	37.8%
Wrong	2	0.1%

4.2. Benchmarking results

As stated in section 1, the original intention was to benchmark with a system of 1,000,000 atoms – namely the Satellite Tobacco Mosaic Virus (STMV). However, due to the fact that memory reduction code described in section 3 was not completed in time, and the relatively limited amount of CPU time available to the project (based on initial estimates, a single SCF iteration of this system would use nearly half of the project's allowance), we chose instead to benchmark a smaller system. The H2O-DFT input file describes a periodically repeating box of 32 water molecules, where this cell is repeated NREP times in each dimension. By varying NREP, test systems of different sizes can easily be constructed. The energy of the system is evaluated using the highly accurate MOLOPT basis set [3] and the PADE exchange-correlation functional.

The benchmarking used the CP2K MPI version executable and tests scaled from 1024 to 8192 processors on JUGENE using the SMP mode. The following figures show the benchmarking performance and scaling cost. Figure 3 shows the timing results of CP2K running with the H2O-DFT input on Jugene. Figure 4 is the scaling cost for the benchmarking. The value of scaling cost is the processor number multiplied by the execution time. A horizontal curve implies linear scaling.

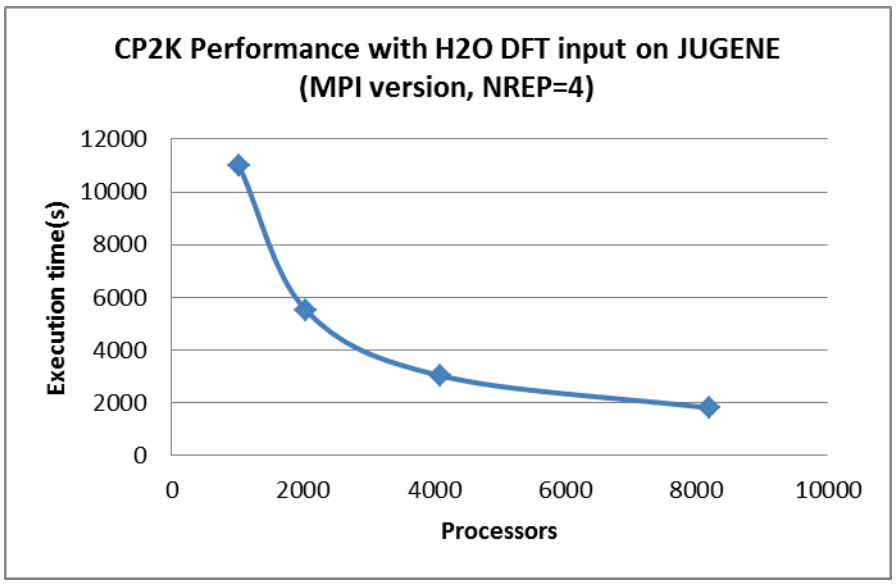


Figure 3 CP2K MPI version benchmarking timing results on JUGENE, with H2O DFT input NREP=4.

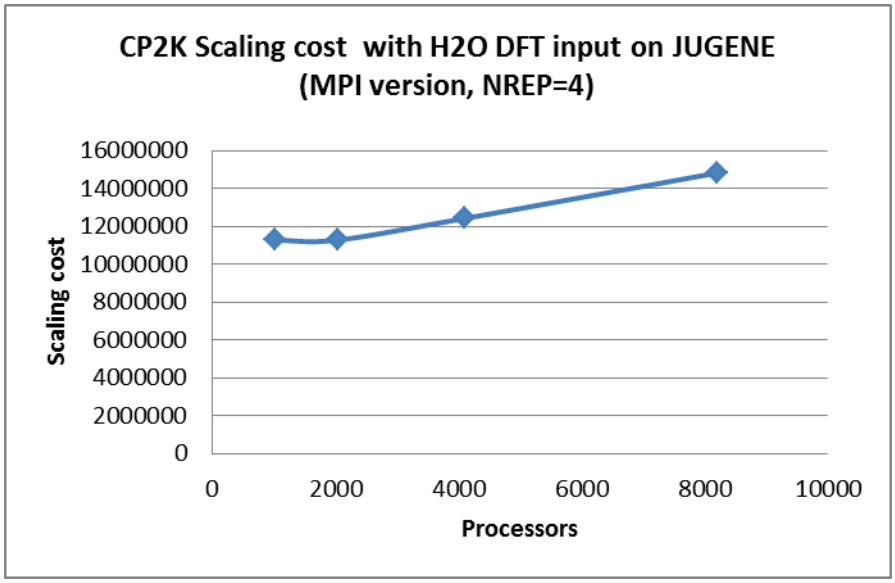


Figure 4 CP2K MPI version benchmarking scaling cost on JUGENE, with H2O DFT input NREP=4. Scaling cost = Processor number * execution time.

Some profiling results including the cost of MPI calls and timings of individual subroutines are also included in the output files of the H2O-DFT benchmarking runs. Table 4 shows the top three most expensive MPI calls for the H2O-DFT benchmark on Jugene with NREP=4. It can be seen that the communication cost is small and not the bottleneck for the performance and scaling of CP2K for this test case. However, it is noted that the Allreduce and Wait calls have begun to increase in cost from 4096 to 8192 processes, although still a tiny fraction of the overall runtime.

Table 4 Top three most expensive MPI calls for H2O-DFT benchmark on Jugene (NREP = 4)

	1024	2048	4096	8192
MP_Allreduce	2.52 s	1.17 s	0.66 s	1.04 s
MP_Wait	2.17 s	1.04 s	0.92 s	1.95 s
MP_Alltoall	0.78 s	0.86 s	0.66 s	0.43 s

Table 5 lists the top five most expensive subroutines for the H2O-DFT benchmark on Jugene with NREP = 4 input. The most obvious feature of this data is that almost all the time is spent in the DBCSR multiplication routine `cp_dbcsr_mult_*` which is a key part of the linear scaling DFT iterative algorithm. However, due to the design of the DBCSR library, which is deliberately loosely coupled with CP2K (for example allowing it to be integrated into other applications), profiling data is only available by default at the CP2K-DBCSR interface (`cp_dbcsr*` functions). Therefore this subroutine hides a great deal of detail of what is occurring within the library, including MPI calls, and several subroutines, whose behaviour would be of interest. Unfortunately not enough time is available to gather this data.

Table 5 Top five most expensive subroutines for H2O-DFT benchmark on Jugene (NREP = 4)

	1024		2048		4096		8192	
Total job time	11009.24 s		5512.72 s		3033.99 s		1809.77 s	
CP2K run time	10803.34 s		5495.54 s		2790.11 s		1462.35 s	
cp_dbcsr_mult_NS_NS	10512.06 s	97.30%	5347.552 s	97.31%	2701.518 s	96.82%	1414.34 s	96.72%
cp_dbcsr_frobenius_norm	90.04 s	0.83%	47.46 s	0.86%	28.987 s	1.04%	14.238 s	0.97%
cp_dbcsr_get_occupation	85.03 s	0.79%	39.225 s	0.71%	25.535 s	0.92%	12.114 s	0.83%
calculate_rho_elec	52.99 s	0.49%	26.524 s	0.48%	13.271 s	0.48%	6.647 s	0.45%
integrate_v_rspace	22.71 s	0.21%	11.373 s	0.21%	5.707 s	0.20%	2.846 s	0.19%

The same test case was also benchmarked with the Mixed-mode version, but was always killed by signal 9. To find out the possible reason, more investigations are needed if further time allows.

5. Partitioning based on hypergraphs

In addition to the work done under PRACE 1-IP WP 7.1 reported here, a group from the Bilkent University Parallel and Distributed Computing Group, Turkey [4][5], have investigated the possibility of replacing the existing regular matrix decomposition with a method based on hypergraph partitioning under WP 7.5.

As mentioned in the project overview, CP2K contains a new linear-scaling SCF method based on the sparse density matrix. This method uses the new parallel sparse matrix multiplication library, DBCSR.

It is reported that for larger systems (up to 1,000,000 atoms), the dominant bottleneck appears to be sparse matrix multiplication which is based on Cannon's algorithm. Its implementation contains nearest-neighbor communication (shifting matrix blocks around a process ring) and also a small amount of all-to-all-type communication in the preparation step.

They propose three novel models based on hypergraph partitioning for multiplication of two sparse matrices (SpMxM) that localize most of the multiplications so that communication is minimized. To the best of our knowledge, this will be the first work that exploits sparsity pattern of the matrices in SpMxM by utilizing hypergraph partitioning. Since sparsity patterns of the two matrices do not change for approximately 10 to 20 iterations, the preprocessing step will be amortized by the speed up in the SpMxM.

So far, they have worked on their models and on their correctness. Investigation of Cannon's algorithm in CP2K, which is implemented in FORTRAN 95, is ongoing. As soon as a solid understanding of the code is gained, they will start to embed their models into the new parallel sparse matrix multiplication library, DBCSR in consultation with the EPCC team.

6. Conclusions

CP2K has been successfully compiled in both MPI-only and mixed-mode MPI/OpenMP versions on the PRACE BlueGene/P system Jugene. Full regression testing has been performed, but some unresolved problems remain which causes some of these tests to hang. However, a high proportion of the tests for both versions of the code passed successfully. Investigations into the memory requirements for large (~1,000,000 atom) systems have been made, along with code development intended to reduce the memory usage of the large neighbour list data structure. To date, this has not been successful and running such large systems on the BlueGene/P proved challenging. Benchmarking has been done using a somewhat smaller test system, and the results show good strong scaling out to 8192 MPI processes. Clearly more work is required before million atom systems can be run easily at Petascale, but we assert that these initial investigations are encouraging, not only by widening the range of systems on which the behavior of CP2K is documented, but also giving wider exposure to the code within the European HPC developer community.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557.

We are very grateful to Dr. Joost VandeVondele and Prof. Jürg Hutter (Univ. Zurich) for help and advice on code development and for access to the XE6 at CSCS for development and testing.

References

1. CP2K Project Homepage, http://cp2k.berlios.de
2. FZJ-ZAM Blue Gene/L User Infcormation – Determine Memory Allocation, http://www2.fz-juelich.de/jsc/jugene/usage/FAQ/memory_allocation
3. Gaussian basis sets for accurate calculations on molecular systems in gas and condensed phases, J. VandeVondele and J. Hutter, Journal of Chemical Physics, vol. 127 (11), 114105, 2007
4. Multi-level Direct K-way Hypergraph Partitioning with Multiple Constraints and Fixed vertices, C. Aykanat, B.B. Cambazoglu and B. Ucar, Journal of Parallel and Distributed Computing, ; vol. 68, pp 609–625, 2008
5. One-Dimensional Partitioning for Heterogeneous Systems: Theory and Practice, A. Pinar, E.K. Tabak and C. Aykanat, Journal of Parallel and Distributed Computing , vol. 68, pp. 1473–1486, 2008.

Appendix

Table 6 Detailed CP2K regression tests status on JUGENE running with MPI version executable (CP2K 2.2.)

Regression Tests	**Total**	**Passed**	**Failed**	**Missed**	**Notes**
All Tests	***2163***	***1723***	***48***	***392***	***Passed Rate:79.7%;*** ***Failed Rate: 2.2%;*** ***Missed Rate: 18.1%;***
ATOM/regtest-1	103	103			
ATOM/regtest-2	14	14			
DFTB/regtest-nonscc	15	15			
DFTB/regtest-scc	24	24			
EP	3	3			
FARMING/regtest-1	9	8	1		Failed due to signal 16
FE/regtest-1	3	3			
FE/regtest-2	1	1			
FE/regtest-3	2	2			
Fist/regtest	97	97			
Fist/regtest-10	17	17			
Fist/regtest-11	13	9	4		Failed due to signal 16 / signal 6
Fist/regtest-12	51	51			
Fist/regtest-13	8	8			
Fist/regtest-14	22	22			
Fist/regtest-15	8	8			
Fist/regtest-2	24	24			
Fist/regtest-3	30	30			
Fist/regtest-4	46	46			
Fist/regtest-5	33	33			
Fist/regtest-6	27	27			
Fist/regtest-7	79	77	2		Failed due to signal 16
Fist/regtest-8	7	7			
Fist/regtest-9	13	13			
Fist/regtest-excl-G	64	64			
Fist/regtest-excl-R	64	64			
Fist/regtest-gauss	25	25			
Fist/regtest-pol	14	14			
LIBTEST	9	9			
MC/regtest	22	7		15	5 tests are not included in the TEST_DIRS; 10 tests have no output contents
NEB/regtest-1	4	4			
NEB/regtest-2	6	6			
NEB/regtest-3	2	2			
NEB/regtest-4	5	3	2		Failed due to signal 16 / signal 6
optimize_input/regtest-1	4	1	2	1	Failed due to signal 16 / signal 6;

					Missed test is not included in the TEST_DIRS
Pimd	11	11			
QMMM/QS/regtest-1	21	21			
QMMM/QS/regtest-2-erf	17	17			
QMMM/QS/regtest-2-swave	17	17			
QMMM/QS/regtest-3	20	20			
QMMM/QS/regtest-4	7	7			
QMMM/QS/regtest-gapw	4	4			
QMMM/SE/regtest	18	18			
QMMM/SE/regtest_2	9	9			
QS	1	1			
QS/regtest-admm	22	8	14		Failed due to signal 16
QS/regtest-all-electron	3	3			
QS/regtest-bs	4	4			
QS/regtest-chi-1	4	4			
QS/regtest-dft-vdw-corr	17	17			
QS/regtest-dm-ls-scf	28	14	14		Failed due to signal 6
QS/regtest-epr-1	4	4			
QS/regtest-epr-2	3	3			
QS/regtest-gapw	22	22			
QS/regtest-gapw-1	8	8			
QS/regtest-gapw-2	5	5			
QS/regtest-gapw-3	4	4			
QS/regtest-gapw-4	15	15			
QS/regtest-gpw-1	42	42			
QS/regtest-gpw-2	29	29			
QS/regtest-gpw-3	39	39			
QS/regtest-gpw-4	39	39			
QS/regtest-gpw-5	22			22	No output content
QS/regtest-gpw-6	19	19			
QS/regtest-hfx	14	12	2		Failed due to signal 6
QS/regtest-hfx-periodic	8	8			
QS/regtest-hfx-stress	4	4			
QS/regtest-hfx-wfn-fitting	6	1	5		Failed due to signal 16
QS/regtest-hole-funct	10	10			
QS/regtest-hybrid	25	22	2	1	Failed due to signal 16; Missed test is not included in the TEST_DIRS
QS/regtest-linearscaling	3	3			
QS/regtest-lsroks	1	1			
QS/regtest-md-extrap	20	20			
QS/regtest-meta	1	1			
QS/regtest-nmr-1	12	12			
QS/regtest-nmr-2	2	2			
QS/regtest-nmr-3	3	3			
QS/regtest-nmr-4	13	13			

QS/regtest-nmr-5	6	6		
QS/regtest-nmr-6	22	22		
QS/regtest-nmr-uks-1	12	12		
QS/regtest-nonortho	12	7	5	5 tests have no output contents
QS/regtest-ot	22	22		
QS/regtest-ot-1	35	35		
QS/regtest-ot-2	41	41		
QS/regtest-ot-refine	9	9		
QS/regtest-ot-refine-2	28	28		
QS/regtest-ot-refine-3	12	12		
QS/regtest-p-efield	4	4		
QS/regtest-plus_u	14	14		
QS/regtest-rtp	26	26		
QS/regtest-spin-spin-1	6	6		
QS/regtest-stress	4	4		
QS/regtest-sym	321		321	No output content
QS/regtest-tddfpt	11	11		
SCP/regtest-scp	8	8		
SE/regtest	44	44		
SE/regtest-2	55	55		
SE/regtest-3	27		27	No output content

Table 7 Detailed CP2K regression tests status on JUGENE running with Mixed-mode executable

Regression Tests	**Total**	**Passed**	**Failed**	**Missed**	**Wrong**	**Notes**
All Tests	***2163***	***1325***	***18***	***818***	***2***	***Passed Rate: 61.3%***
						Failed Rate: 0.8%
						Missed Rate: 37.8%
						Wrong Rate: 0.1%
ATOM/regtest-1	103	103				
ATOM/regtest-2	14	14				
DFTB/regtest-nonscc	15	15				
DFTB/regtest-scc	24	24				
EP	3			3		No output content
FARMING/regtest-1	9	8	1			Failed due to signal 16
FE/regtest-1	3	3				
FE/regtest-2	1	1				
FE/regtest-3	2	2				
Fist/regtest	97			97		No output content
Fist/regtest-10	17	17				
Fist/regtest-11	13	13				
Fist/regtest-12	51	49	2			Failed due to signal 6

Fist/regtest-13	8	8			
Fist/regtest-14	22	22			
Fist/regtest-15	8	8			
Fist/regtest-2	24	23	1		Failed due to signal 16
Fist/regtest-3	30	30			
Fist/regtest-4	46	46			
Fist/regtest-5	33	33			
Fist/regtest-6	27	27			
Fist/regtest-7	79			79	No output content
Fist/regtest-8	7	7			
Fist/regtest-9	13	13			
Fist/regtest-excl-G	64	64			
Fist/regtest-excl-R	64	64			
Fist/regtest-gauss	25	25			
Fist/regtest-pol	14	14			
LIBTEST	9	9			
MC/regtest	22	10	7	5	7 tests failed due to signal 6; 5 tests have no output content
NEB/regtest-1	4			4	No output content
NEB/regtest-2	6			6	No output content
NEB/regtest-3	2			2	No output content
NEB/regtest-4	5			5	No output content
optimize_input/regtest-1	4	1	3		Failed due to signal 6
Pimd	11			11	No output content
QMMM/QS/regtest-1	21	21			
QMMM/QS/regtest-2-erf	17	17			
QMMM/QS/regtest-2-swave	17	17			
QMMM/QS/regtest-3	20	20			
QMMM/QS/regtest-4	7	7			
QMMM/QS/regtest-gapw	4	4			
QMMM/SE/regtest	18	15	3		Failed due to signal 16
QMMM/SE/regtest_2	9			9	No output content
QS	1	1			
QS/regtest-admm	22			22	No output content
QS/regtest-all-electron	3	3			
QS/regtest-bs	4	4			
QS/regtest-chi-1	4	4			
QS/regtest-dft-vdw-corr	17	17			
QS/regtest-dm-ls-scf	28	28			
QS/regtest-epr-1	4	4			
QS/regtest-epr-2	3	3			
QS/regtest-gapw	22	22			HF_gapw_all_LB.inp.out, relative error: 1.46153448e-13
QS/regtest-gapw-1	8	8			
QS/regtest-gapw-2	5	5			
QS/regtest-gapw-3	4	4			

QS/regtest-gapw-4	15	15				
QS/regtest-gpw-1	42	42				
QS/regtest-gpw-2	29			29		No output content
QS/regtest-gpw-3	39	39				
QS/regtest-gpw-4	39			39		No output content
QS/regtest-gpw-5	22			22		No output content
QS/regtest-gpw-6	19			19		No output content
QS/regtest-hfx	14			14		No output content
QS/regtest-hfx-periodic	8	8				
QS/regtest-hfx-stress	4	3	1			Failed due to signal 6
QS/regtest-hfx-wfn-fitting	6			6		No output content
QS/regtest-hole-funct	10	10				
QS/regtest-hybrid	25			25		No output content
QS/regtest-linearscaling	3	3				
QS/regtest-lsroks	1			1		No output content
QS/regtest-md-extrap	20	20				
QS/regtest-meta	1	1				
QS/regtest-nmr-1	12	12				
QS/regtest-nmr-2	2	2				
QS/regtest-nmr-3	3	3				
QS/regtest-nmr-4	13	13				
QS/regtest-nmr-5	6	6				
QS/regtest-nmr-6	22	22				
QS/regtest-nmr-uks-1	12			12		No output content
QS/regtest-nonortho	12	7		5		5 tests have no output content
QS/regtest-ot	22	22				
QS/regtest-ot-1	35	35				
QS/regtest-ot-2	41	41				
QS/regtest-ot-refine	9	9				
QS/regtest-ot-refine-2	28	28				
QS/regtest-ot-refine-3	12	12				
QS/regtest-p-efield	4	4				
QS/regtest-plus_u	14	14				
QS/regtest-rtp	26	26				
QS/regtest-spin-spin-1	6	6				
QS/regtest-stress	4	4				
QS/regtest-sym	321			321		No output content
QS/regtest-tddfpt	11	11				
SCP/regtest-scp	8	8				
SE/regtest	44	42			2	H2O-MNDO.inp.out, relative error : 1.51292015e-02 ; c2h4.inp.out, relative error : 1.29273109e-02
SE/regtest-2	55			55		No output content
SE/regtest-3	27			27		No output content